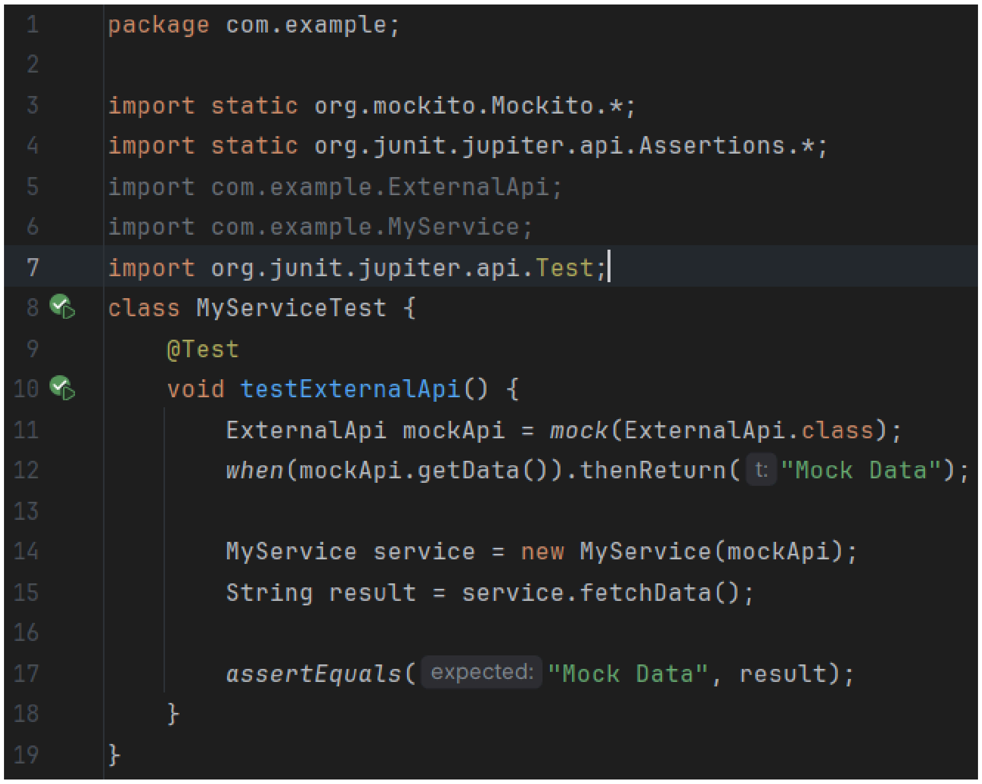
Mockito Hands-On Exercises

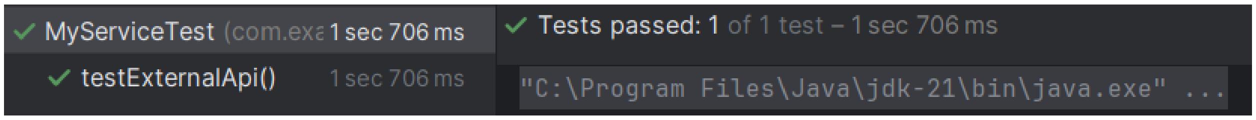
Exercise 1: Mocking and Stubbing

Scenario:
You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:
1. Create a mock object for the external API.
2. Stub the methods to return predefined values.
3. Write a test case that uses the mock object.



```java
package com.example;

import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import com.example.ExternalApi;
import com.example.MyService;
import org.junit.jupiter.api.Test;
class MyServiceTest {
    @Test
    void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn( t: "Mock Data");

        MyService service = new MyService(mockApi);
        String result = service.fetchData();

        assertEquals( expected: "Mock Data", result);
    }
}
```

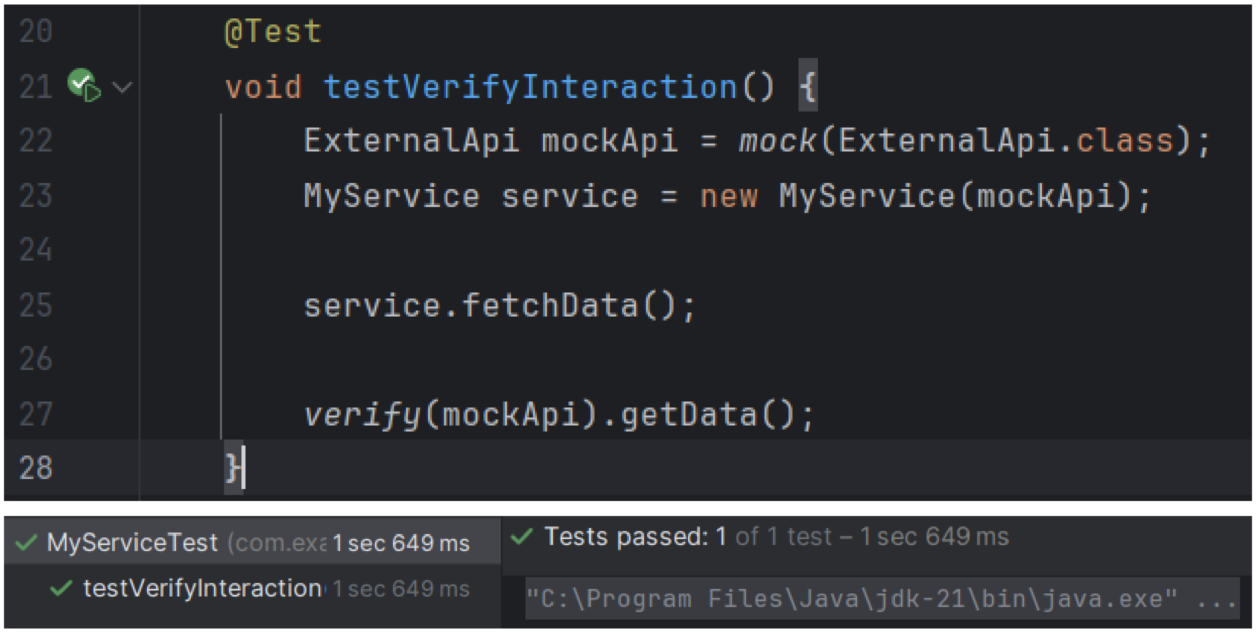


```
✓ MyServiceTest (com.exa 1 sec 706 ms    ✓ Tests passed: 1 of 1 test – 1 sec 706 ms
  ✓ testExternalApi()    1 sec 706 ms    "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

Exercise 2: Verifying Interactions

Scenario:
You need to ensure that a method is called with specific arguments.

Steps:
1. Create a mock object.
2. Call the method with specific arguments.
3. Verify the interaction.



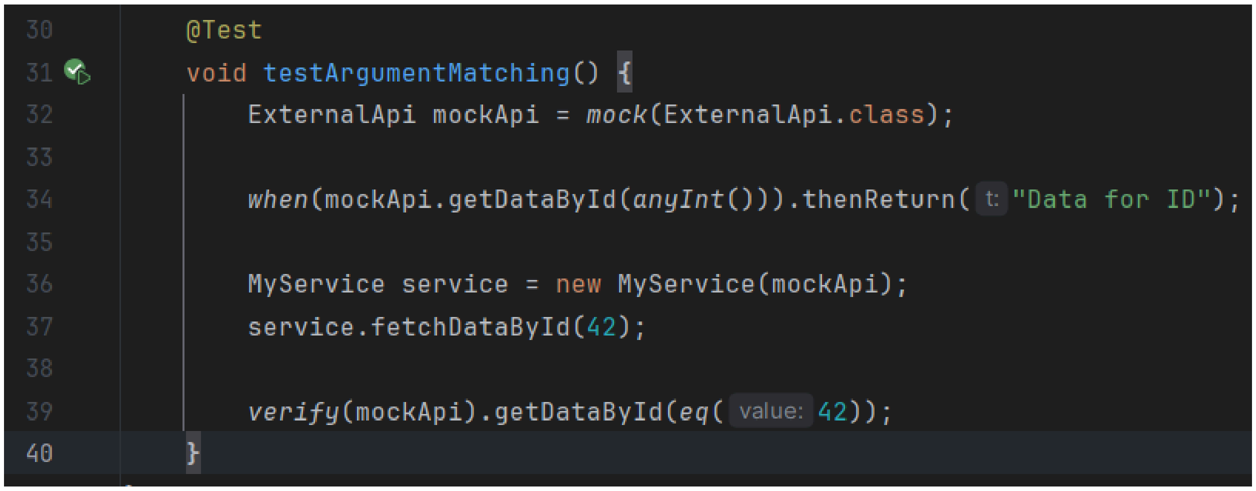
```
    @Test
    void testVerifyInteraction() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService service = new MyService(mockApi);

        service.fetchData();

        verify(mockApi).getData();
    }
```

```
✓ MyServiceTest (com.exa 1 sec 649 ms    ✓ Tests passed: 1 of 1 test – 1 sec 649 ms
   ✓ testVerifyInteraction 1 sec 649 ms   "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

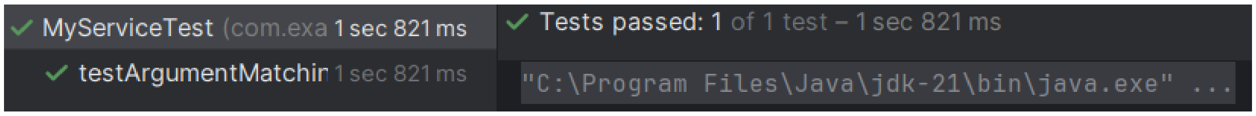
Exercise 3: Argument Matching

Scenario:
You need to verify that a method is called with specific arguments.

Steps:
1. Create a mock object.
2. Call the method with specific arguments.
3. Use argument matchers to verify the interaction.



```java
    @Test
    void testArgumentMatching() {
        ExternalApi mockApi = mock(ExternalApi.class);

        when(mockApi.getDataById(anyInt())).thenReturn( t: "Data for ID");

        MyService service = new MyService(mockApi);
        service.fetchDataById(42);

        verify(mockApi).getDataById(eq( value: 42));
    }
```



```
✓ MyServiceTest (com.exa 1 sec 821 ms    ✓ Tests passed: 1 of 1 test – 1 sec 821 ms
  ✓ testArgumentMatchir 1 sec 821 ms      "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

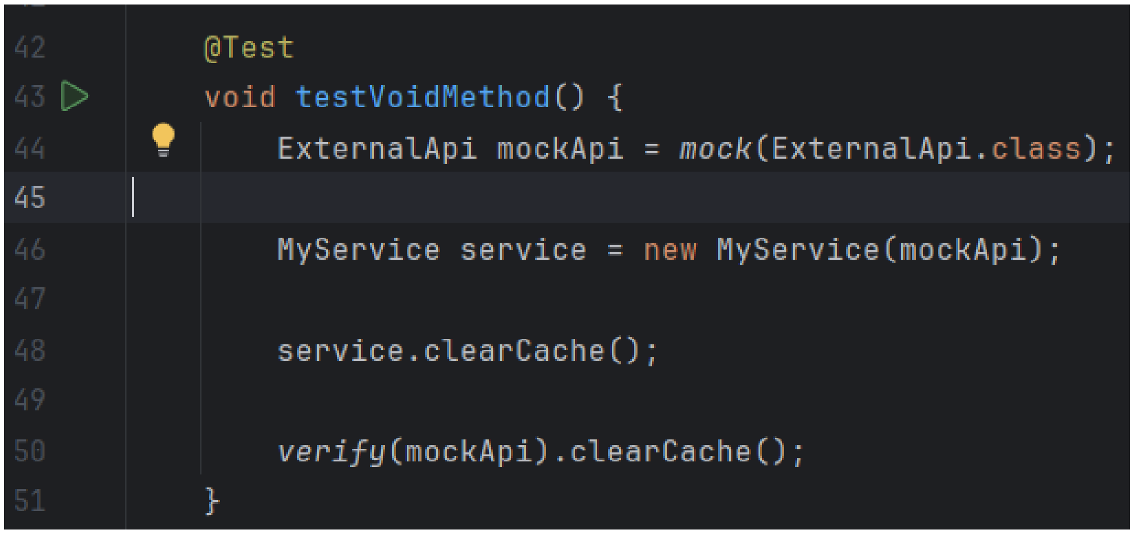
Exercise 4: Handling Void Methods

Scenario:

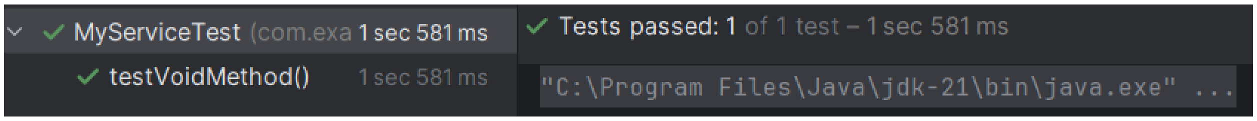
You need to test a void method that performs some action.

Steps:

1. Create a mock object.
2. Stub the void method.
3. Verify the interaction.



```java
    @Test
    void testVoidMethod() {
        ExternalApi mockApi = mock(ExternalApi.class);

        MyService service = new MyService(mockApi);

        service.clearCache();

        verify(mockApi).clearCache();
    }
```



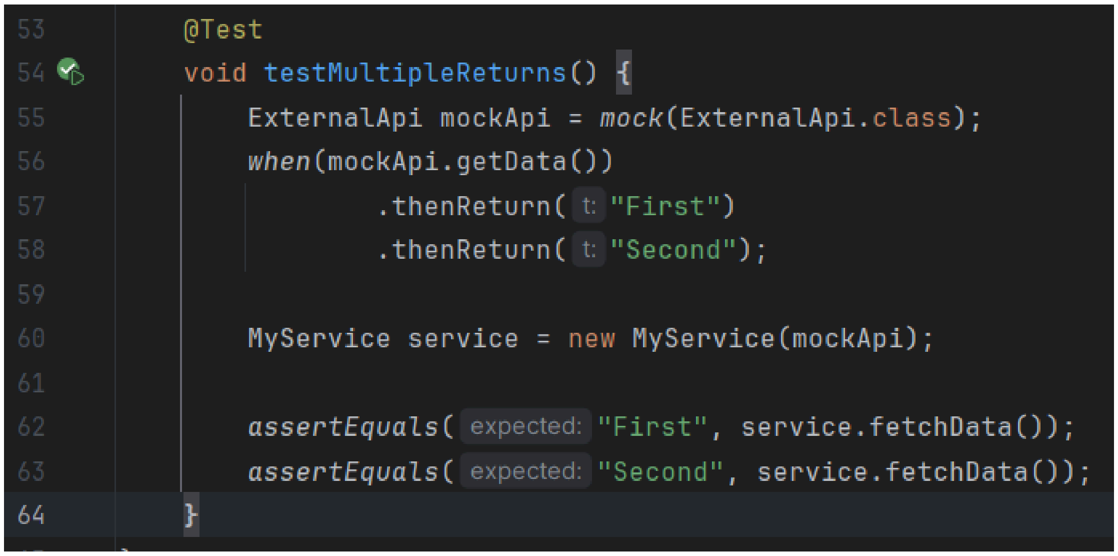
```
✓ MyServiceTest (com.exa 1 sec 581 ms    ✓ Tests passed: 1 of 1 test – 1 sec 581 ms
  ✓ testVoidMethod()      1 sec 581 ms    "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

Exercise 5: Mocking and Stubbing with Multiple Returns

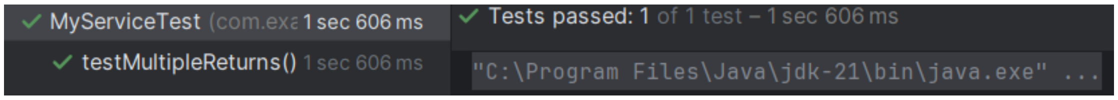
Scenario:
You need to test a service that depends on an external API with multiple return values.

Steps:
1. Create a mock object for the external API.

2. Stub the methods to return different values on consecutive calls.
3. Write a test case that uses the mock object.



```java
    @Test
    void testMultipleReturns() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData())
                .thenReturn( t: "First")
                .thenReturn( t: "Second");

        MyService service = new MyService(mockApi);

        assertEquals( expected: "First", service.fetchData());
        assertEquals( expected: "Second", service.fetchData());
    }
```

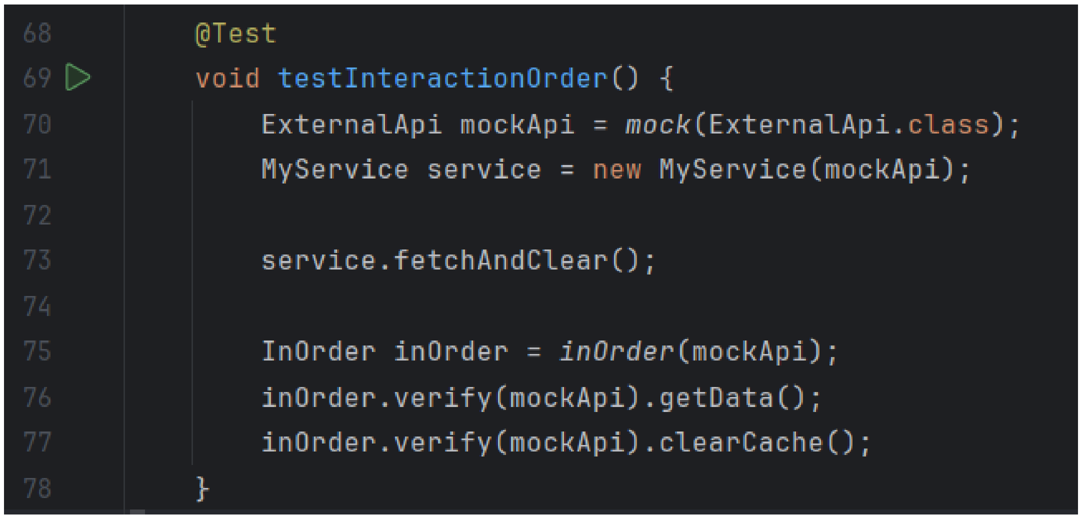


```
✓ MyServiceTest (com.exa 1 sec 606 ms    ✓ Tests passed: 1 of 1 test – 1 sec 606 ms
  ✓ testMultipleReturns() 1 sec 606 ms    "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

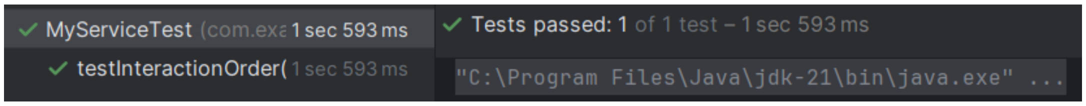
Exercise 6: Verifying Interaction Order

Scenario:
You need to ensure that methods are called in a specific order.

Steps:
1. Create a mock object.
2. Call the methods in a specific order.
3. Verify the interaction order.



```java
    @Test
    void testInteractionOrder() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService service = new MyService(mockApi);

        service.fetchAndClear();

        InOrder inOrder = inOrder(mockApi);
        inOrder.verify(mockApi).getData();
        inOrder.verify(mockApi).clearCache();
    }
```

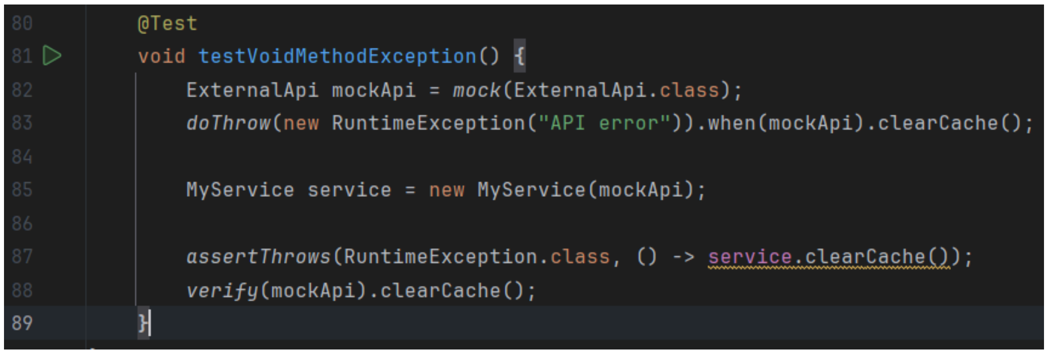


```
✓ MyServiceTest (com.exa 1 sec 593 ms      ✓ Tests passed: 1 of 1 test – 1 sec 593 ms
  ✓ testInteractionOrder( 1 sec 593 ms       "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```

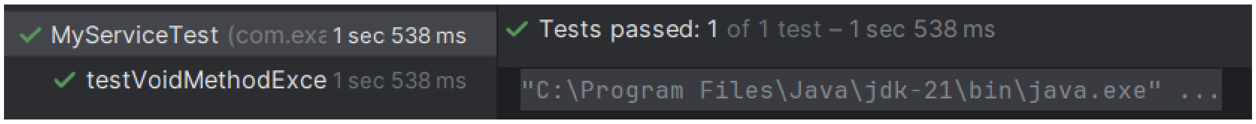
Exercise 7: Handling Void Methods with Exceptions

Scenario:
You need to test a void method that throws an exception.

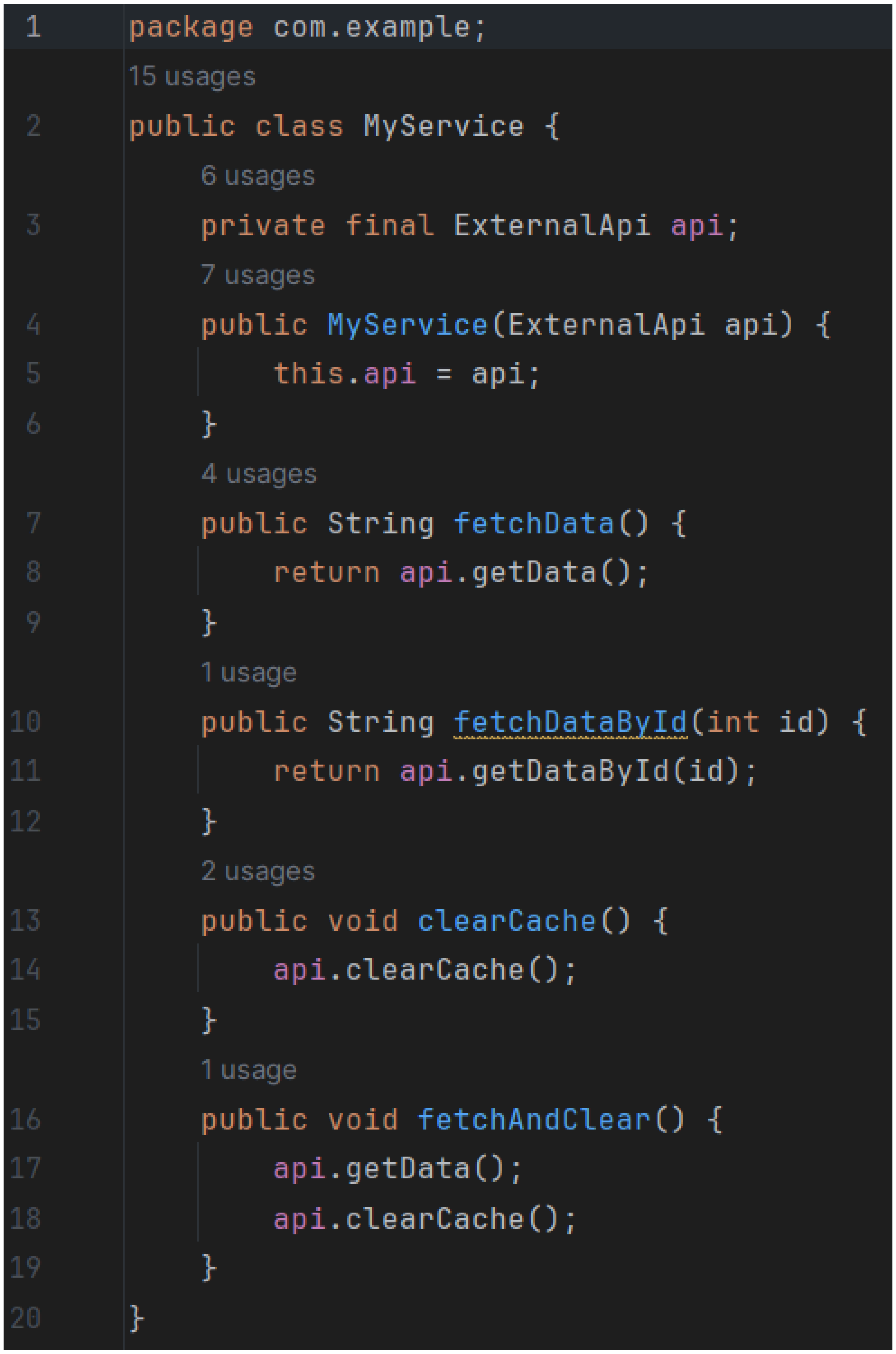
Steps:
1. Create a mock object.
2. Stub the void method to throw an exception.
3. Verify the interaction.



```java
    @Test
    void testVoidMethodException() {
        ExternalApi mockApi = mock(ExternalApi.class);
        doThrow(new RuntimeException("API error")).when(mockApi).clearCache();

        MyService service = new MyService(mockApi);

        assertThrows(RuntimeException.class, () -> service.clearCache());
        verify(mockApi).clearCache();
    }
```



```
✓ MyServiceTest (com.exa 1 sec 538 ms      ✓ Tests passed: 1 of 1 test – 1 sec 538 ms
  ✓ testVoidMethodExce 1 sec 538 ms          "C:\Program Files\Java\jdk-21\bin\java.exe" ...
```



```java
package com.example;
15 usages
public class MyService {
    6 usages
    private final ExternalApi api;
    7 usages
    public MyService(ExternalApi api) {
        this.api = api;
    }
    4 usages
    public String fetchData() {
        return api.getData();
    }
    1 usage
    public String fetchDataById(int id) {
        return api.getDataById(id);
    }
    2 usages
    public void clearCache() {
        api.clearCache();
    }
    1 usage
    public void fetchAndClear() {
        api.getData();
        api.clearCache();
    }
}
```